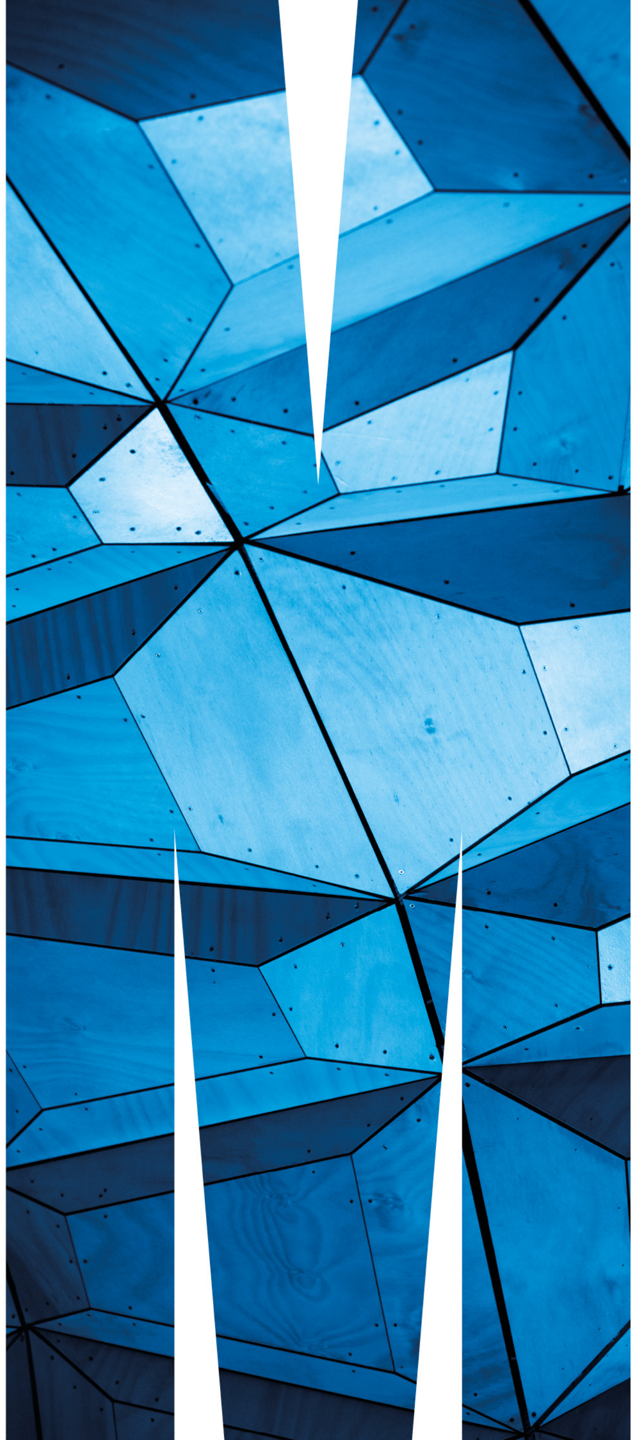
MONASH University

Iterating over Arrays and Execution with Caches

Revisited

Clive Maynard 2020

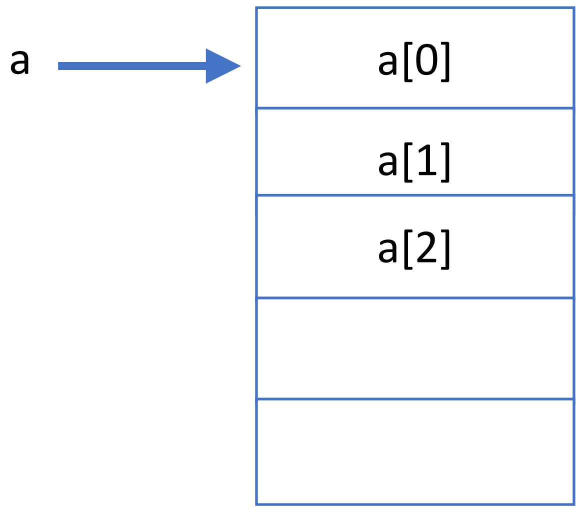


What is this about?

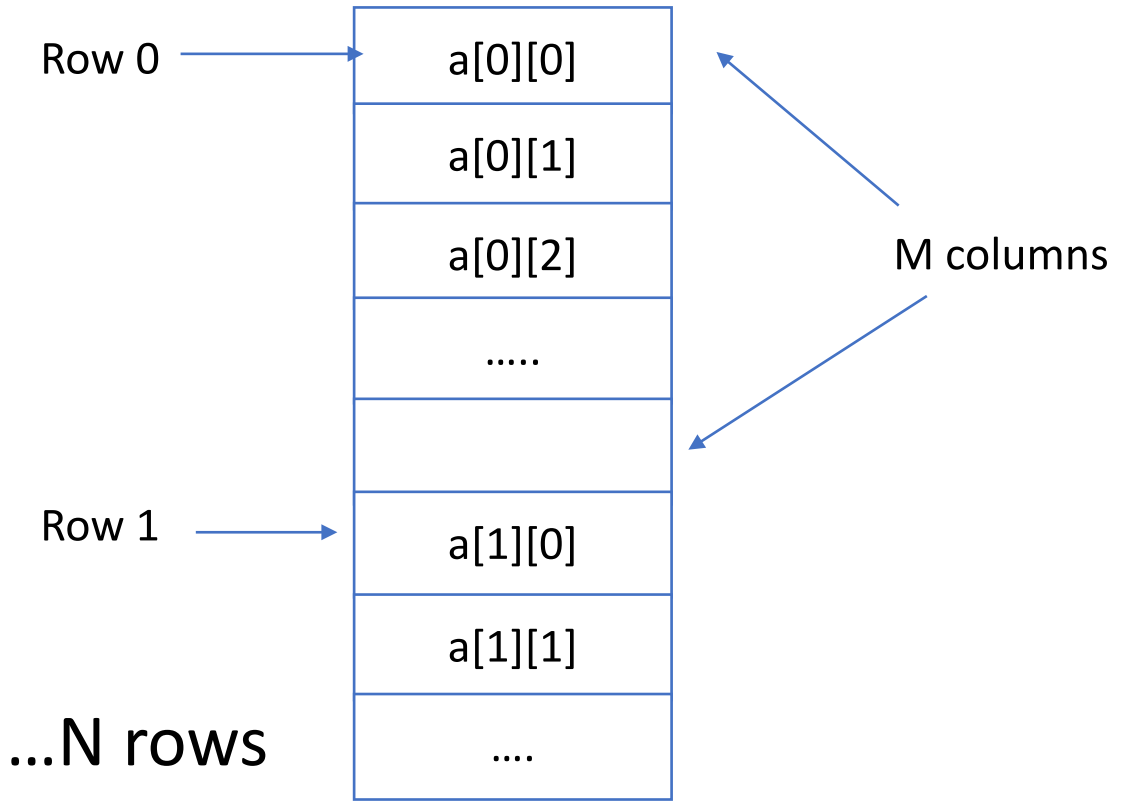
- The original slides have lead to some confusion so I will attempt to revisit the topic and expand on parts of the information to, hopefully, make it easier to understand.
- Some repetition……some new.

Array Storage

1 Dimensional array in C
The array name points to the 0th element



2 Dimensional array in C
The array name points to the 0th element.
There are two distinct storage organisations possible and C uses Row-major.
For a[N][M]



Data access

The difference between the orders lies in which elements of an array are contiguous in memory.
In row-major order, the consecutive elements of a row reside next to each other,
whereas the same holds true for consecutive elements of a column in column-major order.

Some languages use the column-major organisation for example Fortran.

Cache organisation - a reminder

The program wants information from memory and checks the cache.

When a miss occurs the information is loaded from memory into the cache and made available to the program.

Subsequent access to this value gets the cache value (unless it has been replaced between accesses)

To minimise cache misses it is common for the cache controller to automatically load not only the required value but some subsequent values because they will probably be needed soon……remember Spacial Coherence.

How many are loaded is system specific……….for the purposes of this description we will assume the required value and the three subsequent values are automatically loaded.

Think of it this way……..If you are accessing a byte array and the bus is 32bits(4 bytes) wide then the normal operation would be to bring 4 bytes into the cache with no extra system complexity.

A 1D array accessed in a loop

```
for(j = 0; j<8; j++)
    A[j] += 20;
```

The first element is read, modified and stored back and this is repeated for all values in the A[8] array

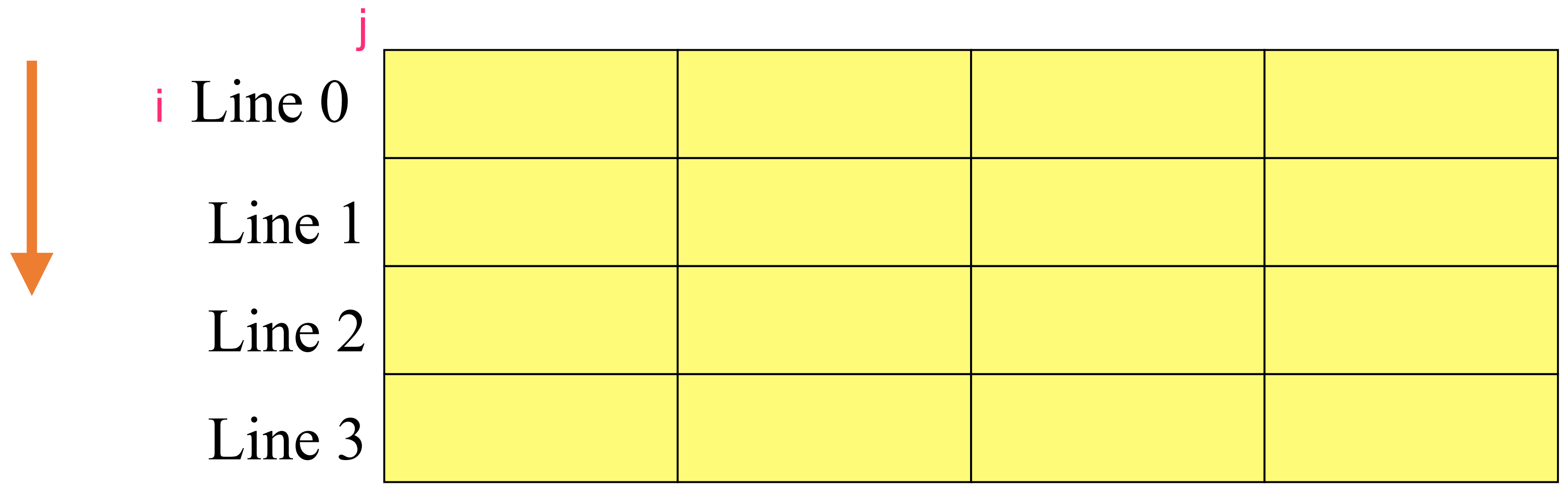
Assuming the cache organisation referred to in the previous slide the execution will be:
Access element 0…..miss and load 4 values.
Access element 1….hit…..and the same for elements 2 and 3
Access element 4….miss and load 4 values.
Access element 5…hit…..and the same for elements 6 and 7

Summarising: 2 misses and 6 hits to access the full array

No problems

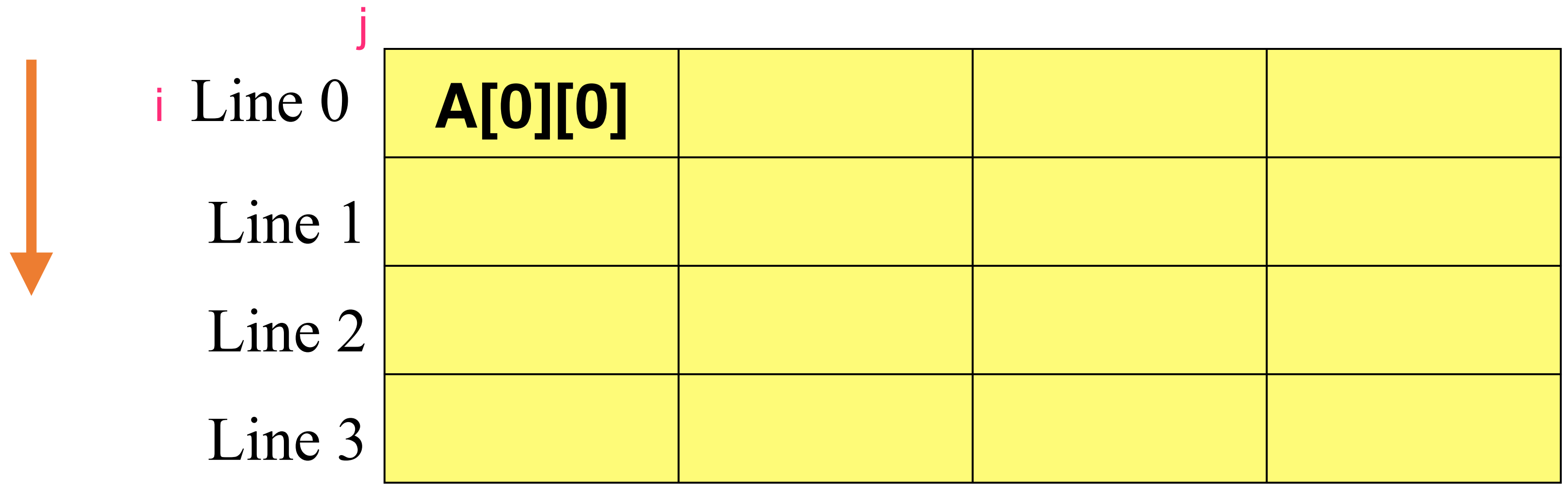
2D array access - A[8][8] - Inner loop changes the row number

```
for(j= 0; j < 8; j++)
        for( i= 0; i < 8; i++)
          A[i][j] += 20 ;
```



2D array access - A[8][8]

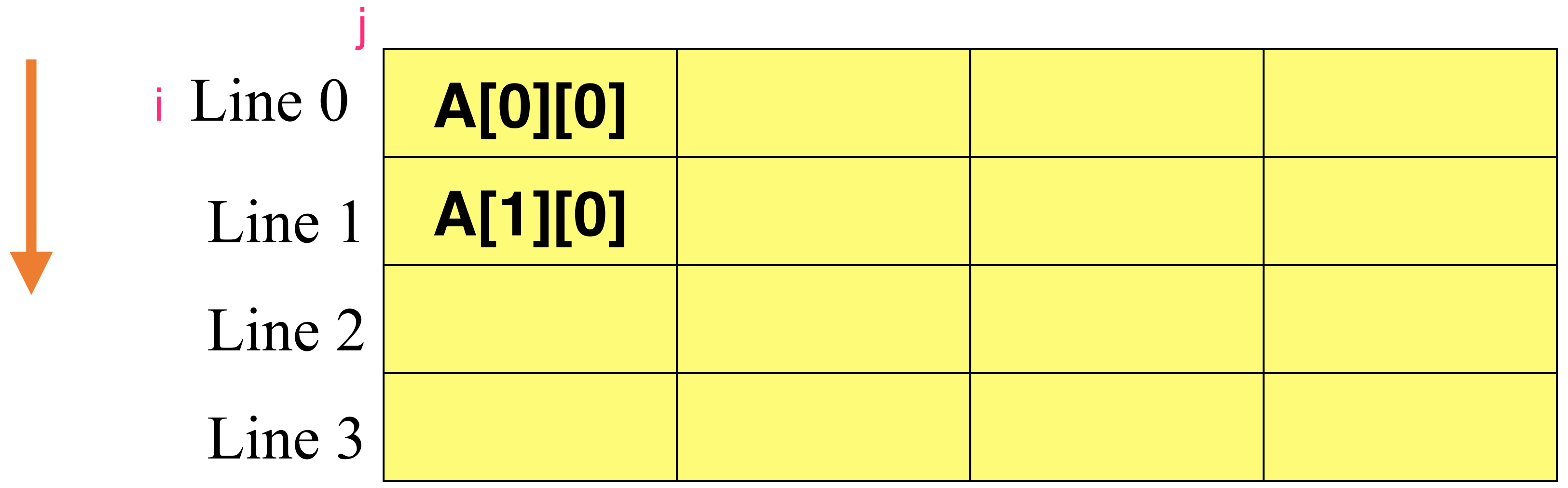
```
for(j= 0; j < 8; j++)
        for( i= 0; i < 8; i++)
          A[i][j] += 20 ;
```



What happens with the cache processing?

2D array access - A[8][8]

```
for(j= 0; j < 8; j++)
        for( i= 0; i < 8; i++)
          A[i][j] += 20 ;
```



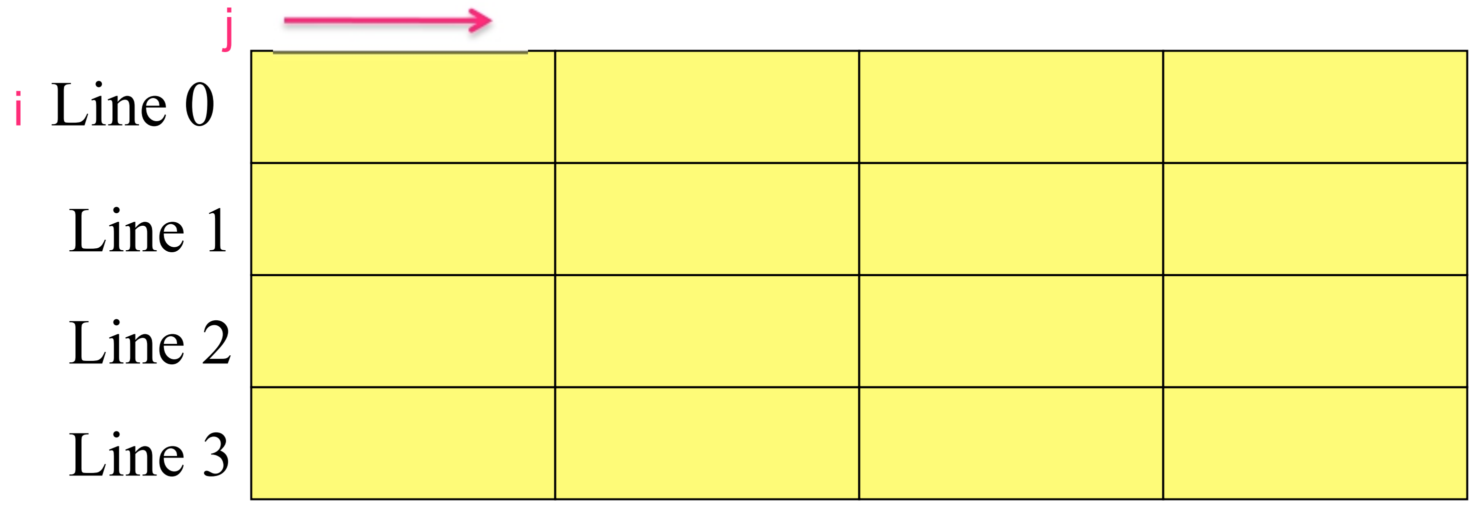
What happens with the cache processing?

Processing with row index change for the inner loop

Assuming the cache organisation referred to in the previous slide the execution will be:
Access element [0][0]…..miss and load 4 values.
Access element [1][0]…..miss and load 4 values
etc
When we get to access element [0][1] this element has probably been displaced from the cache so we are back to….miss and load 4 values.. essentially replacing elements we had already loaded but not used.
Every access is a miss so overall 64 misses with the consequent significant execution time.

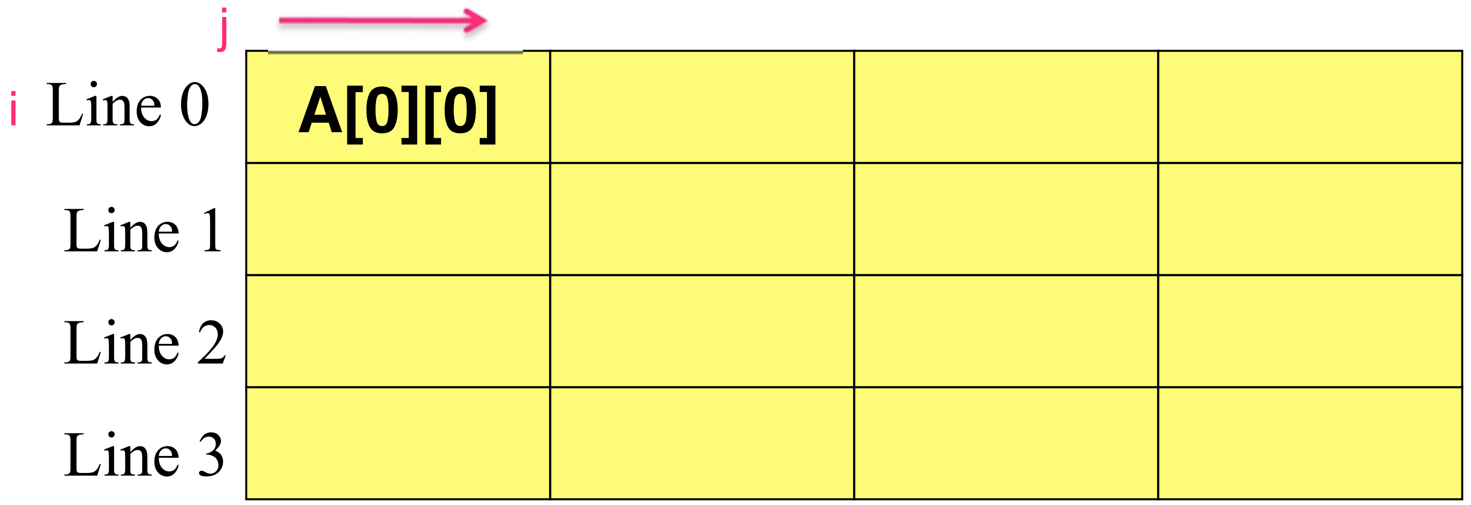
2D array access - A[8][8] - Inner loop changes the column

```
for(i= 0; i < 8; i++)
        for( j = 0; j < 8; j++)
          A[i][j] += 20 ;
```



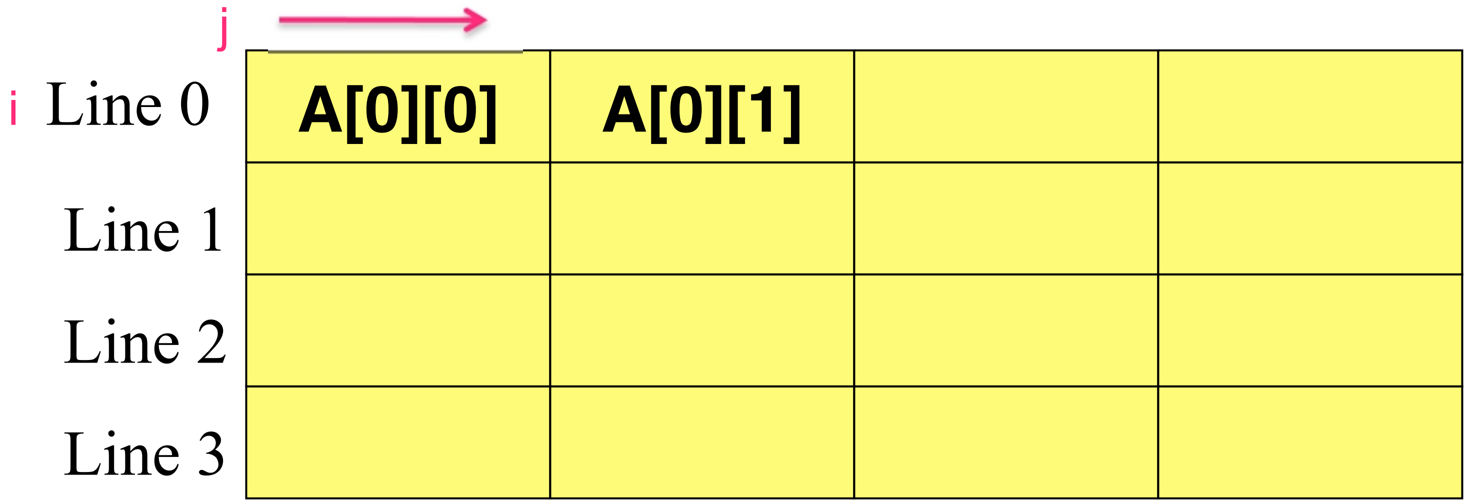
2D array access - A[8][8]

```
for(i= 0; i < 8; i++)
        for( j = 0; j < 8; j++)
          A[i][j] += 20 ;
```



2D array access - A[8][8]

```
for(i= 0; i < 8; i++)
        for( j = 0; j < 8; j++)
          A[i][j] += 20 ;
```



The same access rule and sequencing applies as for the 1D example

The overall cache performance is 16 misses and 48 hits

Summarising

cache access time 10ns, main memory 50ns

Using the memory organisation. Inner loop processes sequential columns of a row

16 misses and 48 hits

1280ns memory access time

Using the memory organisation. Inner loop processes sequential rows of a column

64 misses

3200ns memory access time